UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Anonymity

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

http://www.tlc.unipr.it/veltri

Introduction

- Each time we communicate through a network (e.g. the Internet), we send packets that contain information regarding where the message is going and who sent it (e.g., IP addresses)
  - **anybody observing a link along the path can roughly identify who is communicating with, based on information contained in each packet**
  - **also in case of cryptography is used to protect the integrity and confidentiality of the contents of communication, the source and destination addresses may be still visible to an observer along the route on which a packet is traveling**
  - **this information often is enough to uniquely identify persons participating in a communication**
  - **sometimes these relationships as well as patterns of communication can be as revealing as their content**
- The security service aiming to hide relationships between the communicating parties is called anonymity
- In general, privacy of a communication could require both data confidentiality and anonymity

Anonymity

- Can be defined as “the state of being not identificable within a set of subjects, the anonymity set”
  - **the anonymity set is the set of all possible actors in a system that could have been the sender or recipient of a particular message**
- We could further refine the anonymity set based on the particular role subjects have with respect to a specific message
  - **sender anonymity set**
  - **recipient anonymity set**
- In general, anonymity systems seek to provide "unlinkability"
  - **between sent messages and their true recipients (recipient anonymity), and/or**
  - **between received messages and their true senders (sender anonymity)**

Adversaries – passive/active

- Adversaries of an anonymity system may be:
  - **passive**
    - is able to monitor and record the traffic on network links entering and exiting clients or servers in an anonymity network
    - this can enable powerful statistical attacks
  - **active**
    - has all the monitoring capabilities of a passive adversary
    - is also able to manipulate network traffic by controlling one or more network links or nodes
      - he or she can modify or drop traffic in parts of the network, as well as insert his own traffic or replay legitimate network traffic that he previously recorded

Adversaries – partial/global visibility

- The visibility of an adversary determines how much of the network he or she is able to passively monitor or actively manipulate
- An adversary of an anonymity system may be:
  - **a partial adversary**
    - is only able to monitor the links entering and exiting a particular node in the network, or a small, related subset of nodes in the network
      - an example of a partial adversary might be someone on the same LAN as a node in the anonymity network, or even a common ISP among multiple network nodes
  - **a global adversary**
    - is a powerful observer that has access to all network links in an anonymity network

Adversaries – internal/external

- An adversary may also be
  - **external**
    - does not participate in the anonymity network or its protocol
    - he or she can compromise the communication medium used by clients (i.e., their network links) in order to monitor or manipulate their network traffic
  - **internal**
    - is an active adversary that participates in the anonymity network protocol as a client, or operates a piece of the infrastructure, e.g. by running a node of the anonymity network

Anonymity systems

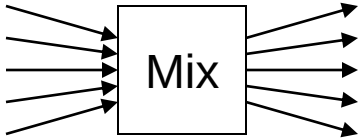
- Tools that attempt to provide anonymous communications, sometimes referred to also as "anonymizers"
  - **Protocol specific anonymizers**
    - implemented to work only with one particular application protocol
    - the advantage is that normally no application extension is needed
      - commands to the anonymizer are included inside a typical message of the given protocol
    - e.g. remailers and web proxies
  - **Protocol independent anonymizers**
    - protocol independence can be achieved by creating a tunnel to an anonymizer
    - protocols used by anonymizer services may include SOCKS, PPTP, or OpenVPN
    - in this case either the desired application must support the tunneling protocol, or a piece of software must be installed to force all connections through the tunnel
      - e.g. web browsers and FTP clients often support SOCKS

High latency vs Low latency

- Anonymity systems can often be classified into two general categories
  - **high-latency systems**
    - high-latency anonymity systems are able to provide strong anonymity, but are typically only applicable for non-interactive applications that can tolerate delays of several hours or more, such as email
    - sometimes referred to as message-based systems
  - **low-latency systems**
    - in contrast, low-latency anonymity systems usually provide better communication performance and are intended for real-time applications, particularly Web browsing
    - sometimes referred to as connection-based systems

High-latency anonymity systems: Mix

- A Mix is the basic building block of nearly all high-latency anonymity systems



- At a high level, it is a process that
  - **accepts encrypted messages as input**
  - **groups several messages together into a batch, and**
  - **decrypts and forwards some or all of the messages in the batch**
- For example, if Alice wants to anonymously send a message M to Bob via a single mix X
  - **Alice sends $E_{Kx}(ID_B,M)$ to X, where**
    - K_X is the public key of X, or a secret key shared between A and X
    - ID_B is Bob's address
    - for preventing an adversary from identifying two identical messages encrypted under the same key, a random value can be also included
      - $E_{Kx}(ID_B,M,R)$
  - **the mix collects messages into a batch until it has received "enough", and then**
  - **forwards each to the destination address extracted from the decrypted input message**

Mix - Flushing algorithm

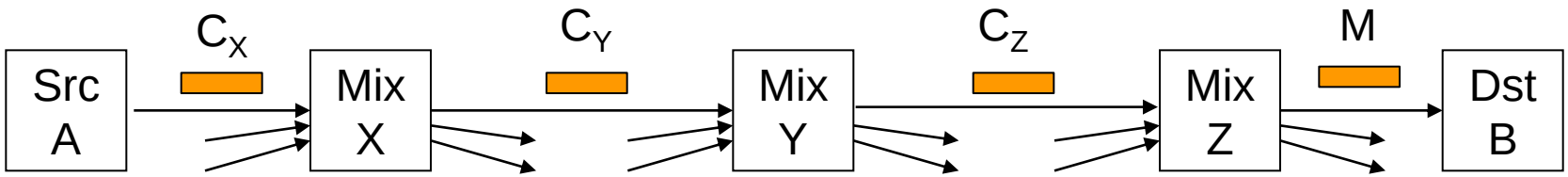
- The algorithm a mix uses to determine which messages to forward to their next destination and when to do so is often referred to as a "flushing algorithm"
  - **sometimes equivalently called "batching strategies"**
- There have been numerous flushing algorithms discussed in the literature, as well as possible active attacks against them
  - **Threshold Mixes**
    - simply collects incoming encrypted messages until it has received *n* messages, then it forwards them to their next destination in a random order
  - **Timed Mixes**
    - collects messages for a fixed length of time Δt
  - **Threshold and Timed Mixes**
    - a combination of the simple threshold and timed flushing algorithms
  - **Pool Mixes**
    - select a random subset of the collected messages to flush and then retain the rest in the mix for the next round
  - **Stop-and-Go Mixes**
    - SG-mixes individually delay messages as they pass through the mix

High-latency anonymity systems: Mix Networks

- Using a single mix leads not only to a single point of failure, but also to a single point of trust since a dishonest mix can reveal the true input-output correlations
- Instead of using a single mix, senders can choose an ordered sequence of mixes through which to send their messages
- If Alice wants to anonymously send a message M to Bob via a path P = {X, Y, Z}
  - **she would iteratively create a layer of encryption, in the same manner as above, for each mix starting with the last mix in the path and working back toward the first:**
    $E_{Kx}(ID_Y, E_{Ky}(ID_Z, E_{Kz}(ID_B, M)))$
  - **Alice then sends the resulting multiply encrypted ciphertext to the first mix in the path**
  - **each mix can remove a layer of encryption to extract the address of the next mix in the path and a ciphertext message to forward to that mix**
- To prevent the lengths of messages from leaking information about how many mixes a message has already passed through, mixes can pad the messages to a fixed length before forwarding them

Mix Networks (cont.)

- Example of Mix Network, with 3 Mixes X, Y, and Z:



- $C_X = E_{Kx}(ID_Y \,||\, C_Y) = E_{Kx}(ID_Y \,||\, E_{Ky}(ID_Z \,||\, E_{Kz}(ID_B \,||\, M)))$
- $C_Y = E_{Ky}(ID_Z \,||\, C_Z)$
- $C_Z = E_{Kz}(ID_B \,||\, M)$

Low-latency anonymity systems

- The flushing algorithms used by the mix-based systems above can introduce large delays on the order of several hours or even days between the time a message is sent and when it arrives at the intended recipient
  - **for applications like email, such delays are tolerable and often even expected**
  - **however, real-time, interactive applications, like Web browsing and SSH, require significantly lower latency**
- This lead to consider low-latency systems
  - **the improved performance of low-latency systems, make these systems more vulnerable to certain traffic analysis attacks**
  - **some types of low-latency systems:**
    - Anonymous Proxy
    - Anonymity Networks
      - Onion Routing

Anonymous Proxy

- Low-latency anonymity systems are often based on the notion of a "Proxy"
  - **in contrast to high-latency anonymity systems (based on mixes)**
  - **while mixes explicitly batch and reorder incoming messages, proxies simply forward all incoming traffic (e.g., a TCP connection) immediately without any packet reordering**
- It is a proxy server that acts as an intermediary between a client and the actual target server
  - **if the communication between the client and the proxy server is encrypted (e.g. via TLS) the client address is then only shared with the proxy server while the target server only sees the proxy server's address**
- Examples
  - **IP VPNs (e.g. OpenVPN)**
  - **HTTPS proxy servers**
- Proxies can also be used in chains forming a sort of Anonymity Networks

Onion Routing

- The most prevalent design for low-latency anonymous communications
- There is a set $R = \{R1,R2, \ldots ,Rn\}$ of servers called Onion Routers (ORs) that relay traffic for clients
- The first step is the establishment of an anonymous connection formed by multiply encrypted tunnel, or circuit, through the network
  - **sequence of *k* ORs**
  - **each OR stores encryption keys, connection IDs, prev/next OR IDs**
- The initiator first selects an ordered sequence of *k* ORs in the network to use as the circuit's path, much like in a mix network
  - **to minimize the computational costs, senders use public key cryptography to establish the circuit and then use faster symmetric key cryptography to transfer the actual data**
  - **the initiator generates two symmetric keys for each OR along the path**
    - a forward key KF_i used to encrypt data sent from the initiator towards the responder
    - a backward key KB_i applied to data from the responder to the initiator

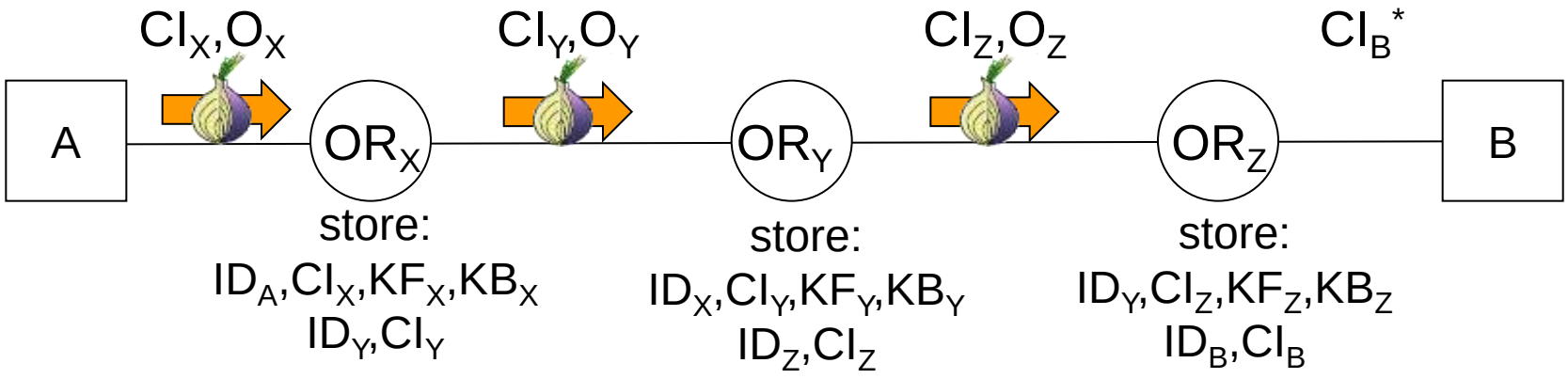
OR Circuit setup

- As an example, if the initiator chose the path $P = \{R_x, R_y, R_z\}$, it would construct the encrypted "onion":
  - $\mathbf{E_{Kx}(t_x,KF_x,KB_x,R_y,E_{Ky}(t_y,KF_y,KB_y,R_z,E_{Kz}(t_z,KF_z,KB_z,ID_B^*,Ø)))}$
    - values t_i indicate the expiration time of the onion
    - $Ø$ means no inner onion is present, indicating to R_z that it is the last router in the path
    - optionally, the address of B (ID_B) can be included in the inner onion in order to establish a connection to node B
- The initiator sends the onion, together with a chosen connection ID CI_x, to the first OR in the path, R_x
  - **which removes from the onion the outermost layer of encryption using its private key, and learns the symmetric keys KF_x and KB_x generated by the initiator, as well as the next server in the path**
  - **R_x then pads the remaining encrypted payload with random bytes, so the onion maintains a constant length, and sends the result, together with a new connection ID CI_y, to R_y**

OR Circuit setup (cont.)

- Each OR along the path:
  - **repeats this process, until the onion reaches the end of the path**
  - **knows its predecessor and successor but no other node in the circuit**
  - **store in a local DB the two connection IDs, the two neighbors' addresses, and the two keys**

- Example:
  - $O_X = E_{Kx}(KF_x \,||\, KB_x \,||\, R_Y \,||\, O_Y)$
  - $O_Y = E_{Ky}(KF_Y \,||\, KB_Y \,||\, R_Z \,||\, O_Z)$
  - $O_Z = E_{Kz}(KF_Z \,||\, KB_Z \,||\, B\ )$



Onion Routing (cont.)

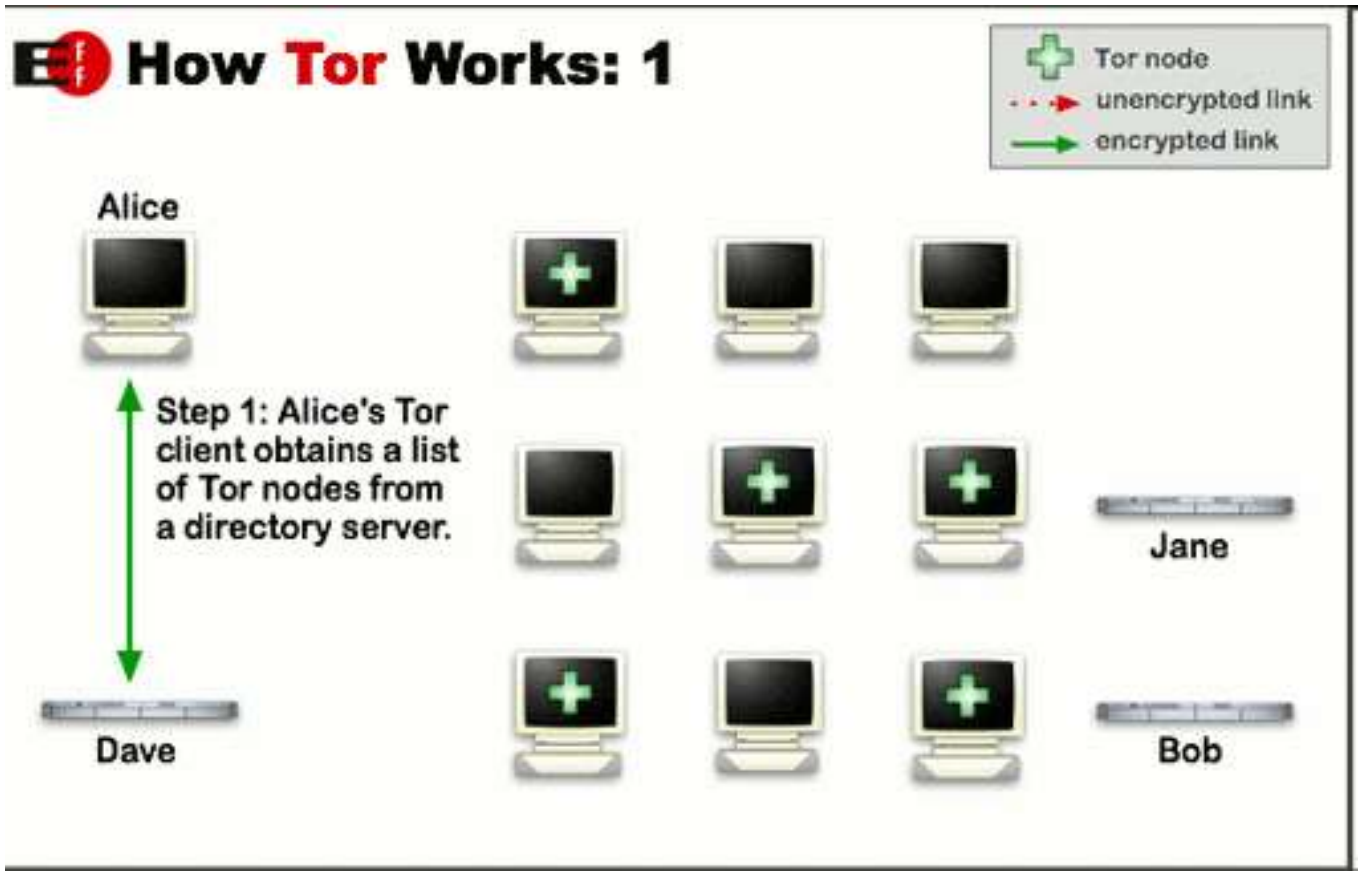
- Once the circuit is constructed, the initiator can relay its application traffic over the circuit using the symmetric keys generated for each hop
- Similarly to how the onion was constructed, the initiator
  - **encrypts a message *m* once for each router in the circuit using the generated forward keys KF_i and then**
  - **sends it, together with the first CI_1, to the first router in the circuit**
  - **data sent from A to the first OR R_1 is: CI_1, $E_{KF1}(E_{KF2}(E_{KF3}(E_{KFn}(m)))$**
- Each OR R_i in the circuit can remove a layer of encryption and forward the result, together with the corresponding output CI_{i+1}, to the next OR_{i+1}, until it reaches the last OR, which can then forward the original message *m* to the destination
- When a response is sent along the reverse path, a layer of encryption is added by each OR in the circuit using the backward key KB_i
- The initiator can then remove all layers of encryption to recover the original response, since he or she has the secret keys that she or he previously generated for each OR

Tor

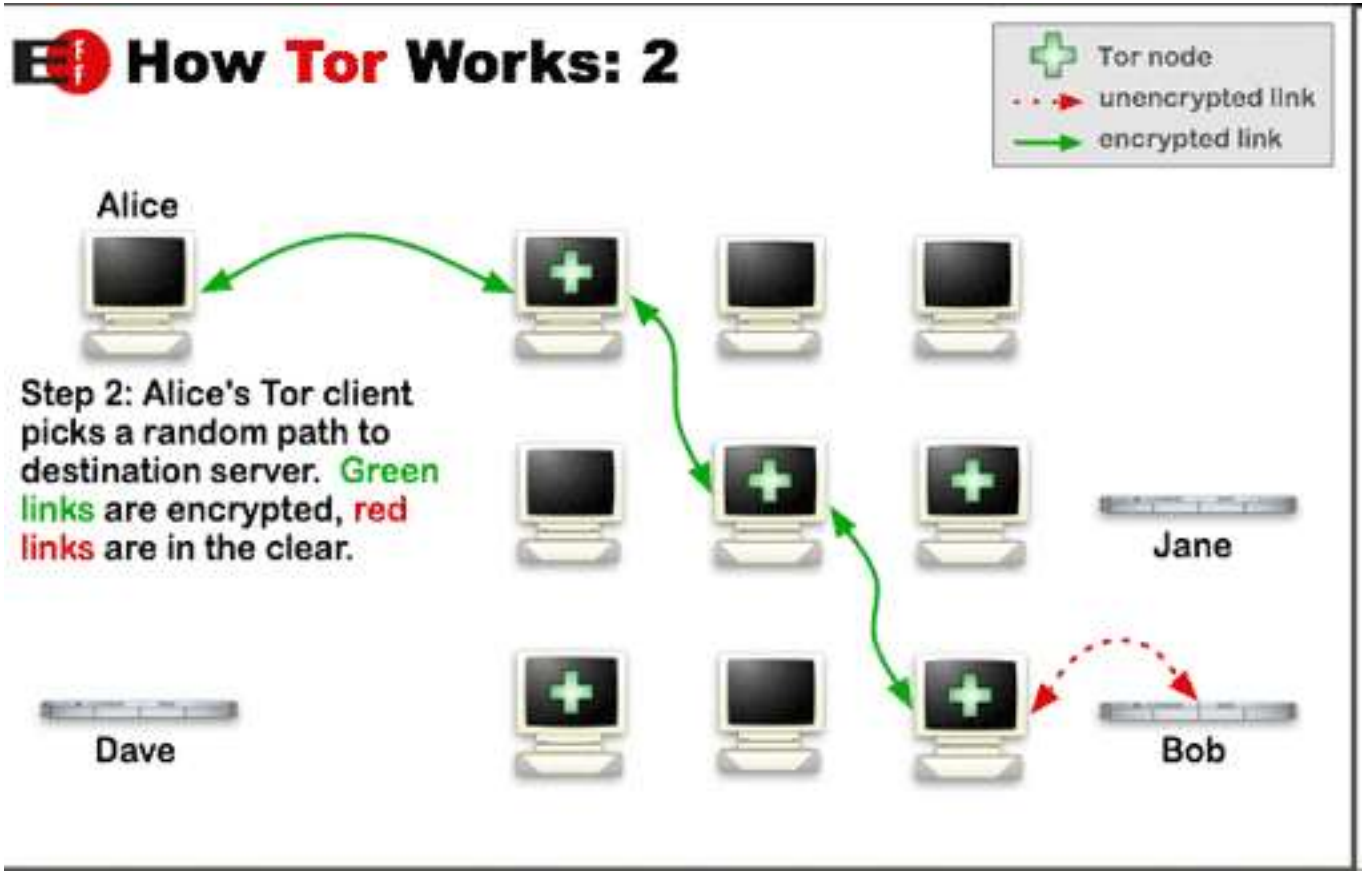
Tor

- Tor (The Onion Router) is a distributed overlay network designed to anonymize low-latency TCP-based applications such as web browsing, secure shell, and instant messaging
  - **originally developed by the U.S. Naval Research Laboratory**
- (Main) Onion routing system
  - **Tor clients running an onion proxy (OP) choose a path through the Tor network and build a "circuit"**
    - the OP periodically negotiates the virtual circuit using multi-layer encryption, ensuring perfect forward secrecy
    - each node (OR) in the path knows its predecessor and successor, but no other nodes in the circuit
  - **the OP presents a SOCKS interface to its clients**
    - SOCKS-aware applications may be pointed at Tor, which then multiplexes the traffic (TCP streams) through a Tor virtual circuit
  - **traffic flowing down the circuit is sent in fixed-size "cells", which are unwrapped by a symmetric key at each node and relayed downstream**

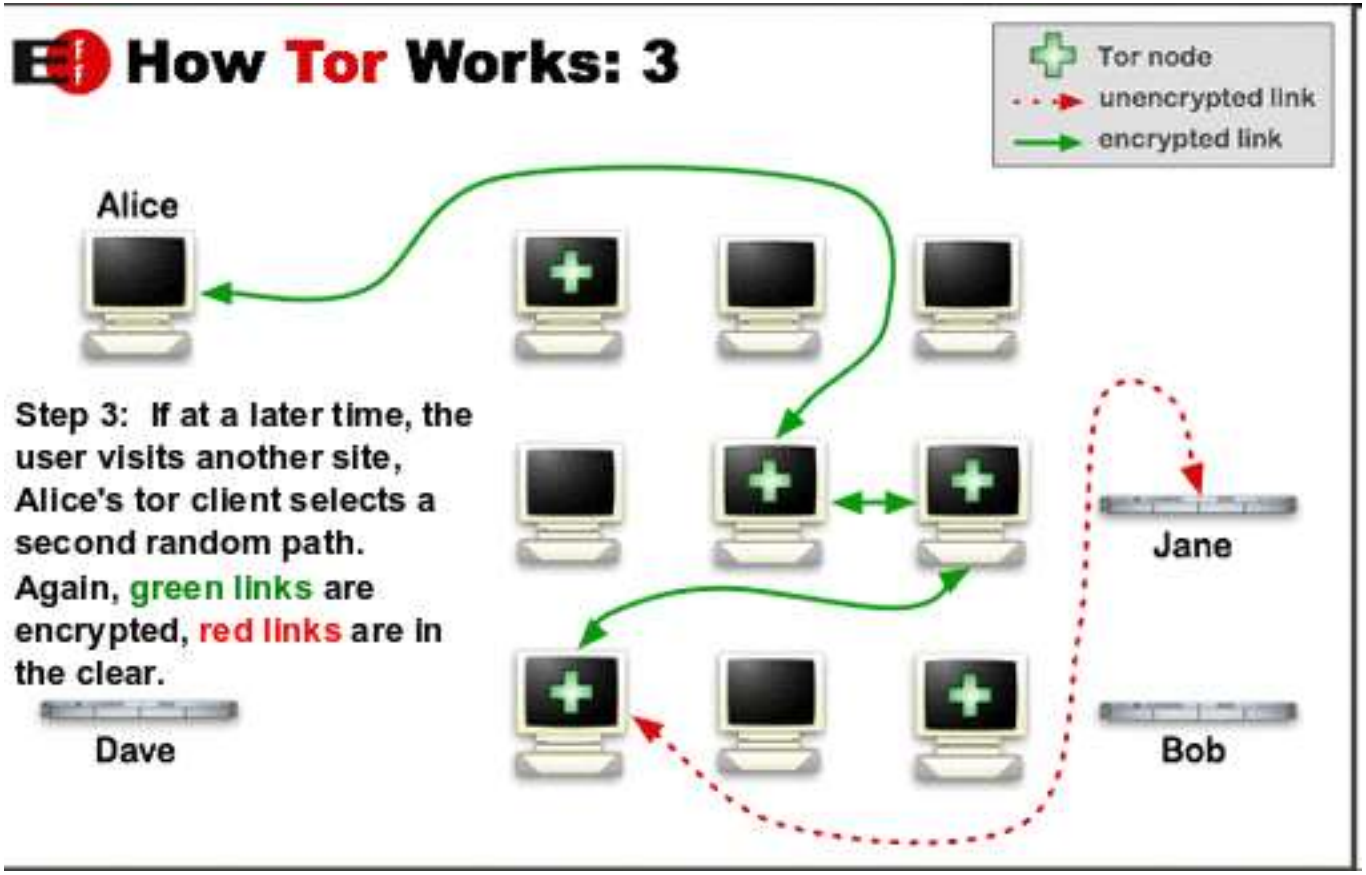
How Tor works



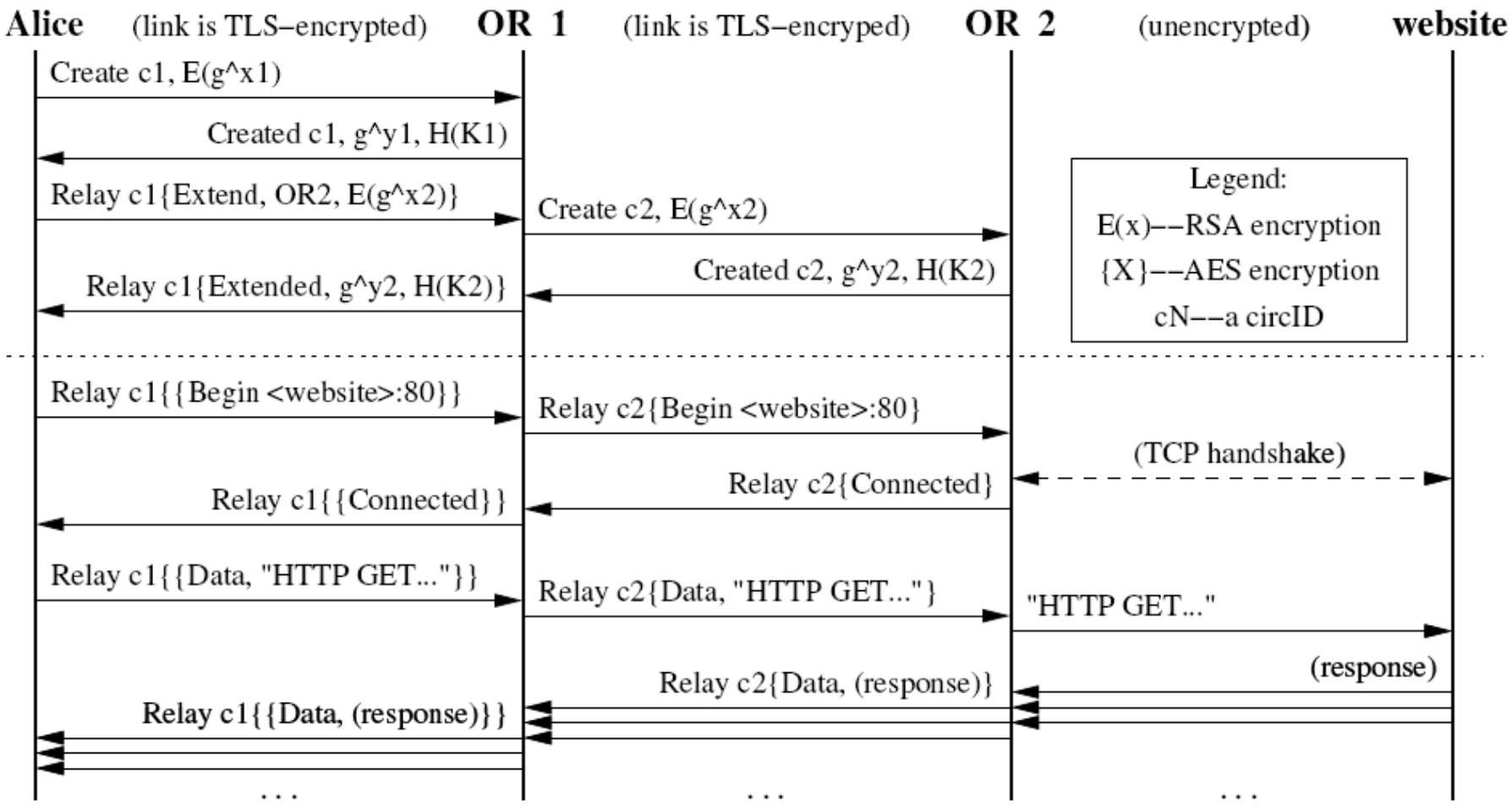
How Tor works (cont.)



How Tor works (cont.)



Tor circuits and streams



Tor circuits and streams (cont.)

- A user's OP constructs circuits incrementally, negotiating a symmetric key with each OR on the circuit, one hop at a time
  - **to begin creating a new circuit, the OP (call her Alice) sends a `create` cell to the first node in her chosen path (call him OR1)**
    - Alice chooses a circID C_1 for the new circuit
  - **the `create` cell's payload contains the first half of the Diffie-Hellman handshake g^{x1}, encrypted to the onion key of OR1**
  - **OR1 responds with a `created` cell containing g^{y1} along with a hash of the negotiated key $K_1=g^{x1y1}$**
- To extend the circuit further:
  - **Alice sends a `relay extend` cell to OR1, specifying the address of the next OR (OR2), and an encrypted g^{x_2} for OR2**
  - **OR1 copies the half-handshake into a `create` cell, and passes it to OR2 to extend the circuit**
    - OR1 associates C_1 to a new chosen circID C_2
  - **when OR2 responds with a `created` cell, OR1 wraps the payload into a `relay extended` cell and passes it back to Alice**
  - **now the circuit is extended to OR2, and Alice and OR2 share a common key $K_2 = g^{x_2y_2}$**

Tor circuits and streams (cont.)

- To extend the circuit to other node or beyond, Alice proceeds as above, always telling the last node to extend one hop further
- Circuit-level handshake and key agreement:
  - **Alice → OR_i : $E_{K^+_{OR1}}(g^{xi})$**
  - **OR_i → Alice : g^{yi},H(K_i|"handshake")**
- This protocol achieves:
  - **unilateral entity authentication**
    - in the second step, OR_i proves that it was he who received g^{xi}, and who chose *yi*
      - Alice knows she's handshaking with the OR_i, but the OR doesn't care who is opening the circuit
        - » Alice uses no public key and remains anonymous
  - **unilateral key authentication**
    - Alice and OR_i agree on a key, and Alice knows only the OR learns it
  - **forward secrecy and key freshness**

Tor circuits and streams (cont.)

- Once Alice has established the circuit (so she shares keys with each OR on the circuit), she can send relay cells
- Upon receiving a relay cell, an OR looks up the corresponding circuit, and decrypts the relay header and payload with the session key for that circuit

Tor circuits and streams (cont.)

- When Alice's application wants a TCP connection to a given address and port, it asks the OP (via SOCKS) to make the connection
  - **the OP chooses the newest open circuit (or creates one if needed), and chooses a suitable OR on that circuit to be the exit node (usually the last node)**
  - **the OP then opens the stream by sending a `RELAY Begin` cell to the exit node, using a new random streamID**
  - **once the exit node connects to the remote host, it responds with a `RELAY Connected` cell**
  - **upon receipt, the OP sends a SOCKS reply to notify the application of its success**
- The OP now accepts data from the application's TCP stream, packaging it into relay data cells and sending those cells along the circuit to the chosen OR

Tor: Hidden services

- Tor can also provide anonymity to websites and other servers
  - **these servers are configured to receive inbound connections through Tor**
  - **they are called hidden services**
- Because hidden services do not use exit nodes, they are not subject to exit node eavesdropping